JOURNAL OF COMPUTER AND SYSTEM SCIENCES 36, 303–312 (1988)

A Measure of Relativized Space Which Is Faithful with Respect to Depth

CHRISTOPHER B. WILSON

Department of Computer and Information Science, University of Oregon, Eugene, Oregon 97403

Received August 29, 1986; revised March 6, 1987

It is known that $NC_1 \subseteq L \subseteq NL \subseteq NC_2 \subseteq NC$; however, known relativizations of these classes have failed to preserve these containments. These methods have failed to preserve either $NC_1 \subseteq L$ or $NL \subseteq NC$ in the sense that one could construct oracles witnessing noncontainment. In this paper we introduce a new measure of relativized space: the oracle stack. With this measure, $NC_1 \subseteq L$ holds for all oracles, as does $NL \subseteq NC_3$ (and $NL \subseteq NC_2$ under certain circumstances). © 1988 Academic Press, Inc.

1. INTRODUCTION

Although the method of relativization has been applied quite successfully to classes defined in terms of Turing machine time [2, 3, 19], an appropriate definition of relativized Turing machine space has been difficult to derive. Simon [16] examines many different definitions, but the two methods that have initially attracted the most attention were characterized by how the space bound was applied to the query tape. One could either subject the query tape to the space bound or exclude it. The former may initially appear to be the most natural; though for certain A, $A \notin L^A$, clearly not a natural situation (see also [1]). On the other hand, if the query tape is excluded from the space bound, then for some A, $NL^A \not\subseteq P^A$ [9]. Furthermore, Savitch's theorem [15] fails to relativize. Our goal is to find a measure of relativized space which retains as many known containments as possible.

Ruzzo, Simon, and Tompa made an interesting step in this direction in [14]. There was introduced what we will refer to as the RST restriction: a nondeterministic Turing machine must act deterministically unless the query tape is empty. However, the query tape is excluded from the tape bound. This restriction attended to the objections raised above, but problems are encountered when comparing space classes defined this way to their natural parallel analog: classes defined in terms of depth [4]. We will state the problems after a few definitions.

The class SPACE($s(n)$) consists of those sets accepted by a Turing machine in $O(s(n))$ space. If we were to refer to functions rather than sets, the output will be written on a write-only tape excluded from the space bound. NSPACE($s(n)$) contains those sets accepted by a nondeterministic Turing machine in $O(s(n))$

0022-0000/88 $3.00

space. We say L is SPACE($\log n$) and NL is NSPACE($\log n$). The function $s(n)$ is called a *space bound* if $s(n) \geqslant \log n$ is a nondecreasing integer valued fully space constructible function.

Our model of parallel computation is the uniform Boolean circuit. In particular we will examine sets accepted by uniform families of circuits whose size and depth are constrained [7, 11]. A set S in $U(u(n)) - \text{SIZE} - \text{DEPTH}(s(n), d(n))$ if there is a family of circuits $\{\alpha_n\}$ such that α_n accepts those strings in S of length n, the size of α_n is at most $O(s(n))$, the depth of α_n is at most $O(d(n))$, and there is a deterministic Turing machine which generates from 1^n an encoding of α_n and uses $O(u(n))$ space. NC_k is defined as $U(\log n) - \text{SIZE} - \text{DEPTH}(n^{O(1)}, \log^k n)$, and NC is the union of all NC_k.

What is currently known is that

$$NC_1 \subseteq L \subseteq NL \subseteq NC_2 \subseteq NC_3 \subseteq \cdots \subseteq NC \subseteq P.$$

None of these containments is known to be strict.

A relativized Turing machine has a query tape upon which it writes a string. When it needs to make a query x to the oracle A, it simply enters a query state with x written on the query tape. In one step, the x is erased and the machine is presented with the answer to "$x \in A$." $\text{SPACE}^A(s(n))$ will be those sets accepted in $O(s(n))$ space relative to the oracle A where the query tape is *not* subject to the space bound. $\text{NSPACE}^A(s(n))$ is defined similarly, but the nondeterministic machine will be *subject to the RST restriction.* An *instantaneous description*, or id, of a Turing machine is a tuple describing the current state of the machine, the locations of the input and worktape heads, and the contents of the worktapes. Notice that if a Turing machine has space bound $s(n)$, then the length of any *id* is $O(s(n))$.

A method to examine relativized circuits was introduced in [17, 18]. To query the oracle, the circuit is allowed oracle gates. These gates take a string x as input and yield as output the answer to $x \in A$. If a gate has k input leads, then it contributes k to the size of the circuit and $\lceil \log k \rceil$ to its depth. A similar idea is used in the notion of NC_1-reducibility [7]. The Turing machine generating the circuit is not allowed access to the oracle. It can be shown that for any oracle A, $NC^A \subseteq P^A$. One can easily construct an oracle B such that

$$NC_1^B = NC_2^B = \cdots = NC^B = P^B.$$

On the other hand, there is an oracle C with

$$NC_1^C \subset NC_2^C \subset \cdots \subset NC^C \subset P^C.$$

2. Oracle Stacks

As mentioned earlier, the fact that $NC_1 \subseteq L$, and, in general, $U(s(n)) - \text{DEPTH}(s(n)) \subseteq \text{SPACE}(s^k(n))$, does not relativize [18]. Indeed, we saw

there an oracle D where, for any k, NC_1^D contains a set not in $\text{NSPACE}^D(n^k)$. The fact that $NC_1 \subseteq L$ does not relativize is due to the fact that an NC_1 circuit essentially has a built in memory of the outcomes of all the queries: the output lines from each of the query nodes provide this. So in a circuit, one query can be constructed from the outcomes of $\omega(\log n)$ previous queries. No log-space machine has the memory to accommodate this information.

The remedy, then, is to allow the space-bounded machines to have more than one query tape, on which partially constructed queries can be stored. We would have to settle on a structure for the collection of tapes and a method, if desired, to measure the space usage. A model along these lines is found in [10], where random access is allowed to the tapes through an index tape. Only the index tape was included in the space bound. This approach is adopted in [5] when discussing a variant (called the m-variant) of relativized space, with the modification that the space be the sum of the logarithms of the lengths of the queries. This is essentially the measure we will use, but the query structure will be different. Here, we will put the partially constructed queries on a stack, as suggested by Cook [6]. This structure takes advantage of the essentially tree-like nature of the circuits.

DEFINITION 1. An *oracle stack* is a push down stack replacing the query tape, where each entry can be thought of as an independent query tape. The machine is allowed to write onto the top entry, possibly adding to what is already there, but is not allowed to read it. When the machine is finished writing, it can either push, at which point the tape is pushed down and the top tape becomes empty, or it can query, which causes the contents of the top tape to be queried and erased and the stack to be popped.

DEFINITION 2. If the contents of the stack at any point in the computation are the partial queries $q_1, q_2, \ldots, q_k$, the *space currently used* by the stack is

$$\sum_{i=1}^{k} \max(\log|q_i|, 1).$$

The *space used* during the computation is the maximum over all steps in the computation of the space currently used by the stack and worktapes.

Notice that a machine whose stack space is bounded by $s(n)$ can query strings of length at most $2^{s(n)}$.

Defining the use of the stack is not a problem for deterministic machines, but for nondeterministic ones there may be several possibilities. It is straightforward for unrestricted nondeterminism: where $s(n)$ is the space bound, for the nondeterministic Turing machine to accept an input of length n, there must be an accepting computation which uses at most $s(n)$ work space and stack space. Under the RST restriction, we must decide at which points the machine must act deterministically. If it is to be between starting to write on the stack until a push or a query, then this turns out to be no restriction at all. The machine could write $s(n)$ nondeter-

ministically chosen bits onto a work tape, copy them onto the stack, push, query a dummy string (causing a pop), copy more nondeterministically chosen bits onto the stack, and so on. So we will force the machine to act deterministically from the point at which it writes onto an empty stack (entering a *beg-write-id*) to the first point at which the stack is again empty

DEFINITION 3. L in in $s\text{SPACE}^A(s(n))$ if and only if it is accepted by a Turing machine relative to A with an oracle stack which uses space at most $O(s(n))$.

DEFINITION 4. L is in $s\text{NSPACE}^A(s(n))$ if and only if it is accepted by a nondeterministic Turing machine using an oracle stack whose space usage is at most $O(s(n))$ under the restriction that the machine may make nondeterministic moves only when the stack is empty.

DEFINITION 5.

$$sL^A = \bigcup_{c \geq 1} s\text{SPACE}^A(c \log n)$$

$$sNL^A = \bigcup_{c \geq 1} s\text{NSPACE}^A(c \log n).$$

Let us compare this model to the m-variant of relativized space introduced by Buss [5]. There a Turing machine may have many query tapes and a single index tape. If i is written on the index tape, then all writes, queries, and answers apply to tape i. At any point, if there are q active queries of maximum length m, the space used is $q \log m$. This is essentially our stack space measure; however, only the top query is available to the Turing machine. In addition, Buss proposes a model of relativized alternation, which itself can be viewed as a model of relativized parallel computation (see [13]). The alternating Turing machine, as well as having existential and universal states, is allowed conditional states where the state of one successor depends on the outcome of the other successor. Hence, the machine can write a query bit which is dependent on further computation and other queries. The results of immediate interest are that for any oracle A,

$$m\text{SPACE}^A(s) \subseteq m\text{ATIME}^A(s^3)$$

and

$$m\text{NSPACE}^A(s) \subseteq m\text{SPACE}^A(s^2).$$

The latter result illustrates that Savitch's theorem relativizes, as it does for the stack model (see Theorem 4). The former result is surprisingly analogous to Theorem 3, though it refers only to deterministic space.

3. Faithful Containments

The major appealing feature of this model is that when simulating a circuit family by a Turing machine, the latter's space does not blow up relative to the former's depth.

THEOREM 1. *For any oracle A and space bound $s(n)$,*

$$U(s(n)) - \text{DEPTH}^A(O(s(n))) \subseteq s\text{SPACE}^A(O(s(n))).$$

Proof. The proof of this is fairly simple. The Turing machine can derive the relevant parts of the circuit in $O(s(n))$ space due to the uniformity condition. As in the unrelativized case in [4], it will perform a depth-first search of the circuit starting at the circuit's output edge. As bits of a query are computed, they are written on a query tape. If determining further bits of the query involves evaluating a subsection of the circuit, then the partial query is pushed and stored while the evaluation continues. Since the sum of the logarithms of the size of the queries on any directed path from an input edge to the output edge is bounded by $O(s(n))$, so bounded will the space used by the stack be. ∎

COROLLARY 2. *For any oracle A, $NC_1^A \subseteq sL^A$.*

So by strengthening the space-bounded Turing machines, we have ensured that $NC_1^A \subseteq sL^A$. What is not clear is whether sNL^A will be contained in NC_2^A, or even NC^A. In [18] there were constructed some sets $L_k(A)$ which, for some A, are not in NC_{k-1}^A, but are always in NC_k^A. These $L_k(A)$ fortunately are not in sL^A or sNL^A, but they would be if the machines could make arbitrarily many copies of the same item in the stack. This could happen if random access (write-only) to the partial queries were allowed, if the query tape were not erased, or if the top query tape was copied into the stack rather than pushed. The similar notion of relativized space used in [10] would allow that any $L_k(A)$ be computed in log-space relative to A.

Under certain circumstances, it is easy to see that $sNL^A \subseteq NC_2^A$. Suppose that all queries were of length at most constant. Then there would be at most a constant number of them, and they could all be queried simultaneously at the top level of the circuit. An NC_2 circuit below this could complete the computation. On the other hand, suppose that the smallest query on the stack has length n. Since the space used is $O(\log n)$, there will be at most a constant number of strings on the stack. So there would only need to be a constant number of query levels, with an NC_2 circuit between each level. A problem, however, arises when the items on the stack are of intermediate length. In that case, we can at least show that sNL will be in NC, and in particular it will be contained in NC_3.

THEOREM 3. *For any oracle A, $sNL^A \subseteq NC_3^A$.*

Proof. For an arbitrary A, let $S \in sNL^A$. Then S is accepted by a nondeterministic Turing machine M with an oracle stack which uses work space and stack space bounded by $O(\log n)$. We will show that there is an NC_3^A circuit which, on an

arbitrary input x of length n, simulates M on x. This circuit, incidentally, is independent of A. By the definition of sNL, the machine M will act deterministically when there are items on the stack. Furthermore, the stack height is at most $c \log n$ for some constant c and the queries are at most polynomial in length. (In fact, we could restrict things a bit more—certainly not all $c \log n$ queries are of polynomial length. Taking advantage of this fact might allow us to show that the language is in NC_2^A.)

The main idea of the proof is to compute the reachability graph on the ids first for stack height 0 (trivial), then for stack height 1, then stack height 2, and so on. Computing this graph for stack height $k+1$ from the one for stack height k will be seen to be an NC_2 circuit. Thus, the total depth will be $c \log n \cdot O(\log^2 n) = O(\log^3 n)$ to compute the connectivity of the ids when something is on the stack. The nondeterministic closure can then be computed by an NC_2 circuit.

We define the following function:

on inputs α, β, and k, where α and β are ids, $|\alpha|, |\beta| \leqslant O(\log n)$, and k is an integer $0 \leqslant k \leqslant c \log n$,

$u(\alpha, \beta, k) = 1$ if and only if M can reach β from α in such a way that

- the stack is empty at ids α and β and
- for all intermediate ids, the stack height is at least one but never more than k.

That is, α begins to write on a tape and β is the direct result of the first pop (query) which empties the stack, which in between never holds more than k partially constructed queries at a time. Also note that $u(\alpha, \beta, 0) = 1$ if and only if α yields β in one step, without recourse to the oracle.

Define U_k to be a bit string whose $\langle \alpha, \beta \rangle$th bit is $u(\alpha, \beta, k)$, where $\langle \cdot, \cdot \rangle$ is an easily computable bijection of $\mathbf{N}^2$ to $\mathbf{N}$. Since the ids have length bounded by $c \log n$, there are polynomially many of them, and so U_k also has polynomial length. It is easy to see that U_0 can be computed by an NC_1 circuit.

Claim. There is an NC_2^A circuit computing U_{k+1} from U_k.

Since the length of each U_k is at most polynomial, the claim will follow from the following subclaim by computing each bit in parallel.

Subclaim. For each id pair α, β, there is an NC_2^A circuit computing the $\langle \alpha, \beta \rangle$th bit of U_{k+1} from U_k. This we show by

(i) determining $u(\alpha, \beta, k+1)$ from U_k assuming the answer to the query yielding β to be both yes and no; that is, compute both $u_{\text{yes}}(\alpha, \beta, k+1)$ and $u_{\text{no}}(\alpha, \beta, k+1)$

(ii) calculate the last query made

(iii) make that query, returning ans $\in$ {yes, no}

(iv) $u(\alpha, \beta, k+1)$ will be $[u_{\text{yes}}(\alpha, \beta, k+1)$ and ans = yes$]$ or $[u_{\text{no}}(\alpha, \beta, k+1)$ and ans = no$]$.

We will see that both steps (i) and (ii) can be computed in logarithmic space, hence both can be calculated by an NC_2 circuit. Step (iii) is clearly in NC_1—a single polynomial size, and hence log depth, query will suffice. Also, step (iv) is constant depth.

We now show that given α, β, U_k, and the original input x, in log-space we can compute $u(\alpha, \beta, k+1)$ with the assumption χ as the response to the last query.

ALGORITHM.

1. If $u(\alpha, \beta, k) = 1$, then output 1 and halt.
2. Ensure that α begins to write to the query tape. If not, then output 0 and halt.
3. Simulate M starting from id α until an id α' is reached which causes either a query (pop) or a push (new beg-write-id). During this simulation, ignore any writing to the query tape.
4. If id α' causes a push, then search for a β' such that bit $\langle \alpha', \beta' \rangle$ of U_k is 1 (that is, $u(\alpha', \beta', k) = 1$). Let α be this β' and return to step 3. If no such β' exists, then output 0.
5. [Here, α' causes a query.] Test if α' yields β with χ as the oracle response.
6. If so, then output 1. Otherwise, output 0.

The algorithm is $O(\log n)$ space, so can be performed by an NC_2 circuit. By a similar argument, we can see that step (ii) of our agenda can also be done in log-space. In the algorithm above, in step 3 during the simulation, instead of ignoring any writing to the query tape, direct them to an output tape. Note that for both steps (i) and (ii), step 3 of the algorithm will be deterministic. This behavior of M is guaranteed by the definition of our oracle stack model.

Thus, the subclaim and therefore the claim hold. So given the input x, we can compute U_K, where $K = c \log|x|$ is the maximum stack height, on an NC_3^A circuit. To complete the proof, we show that given x and U_K, the answer to $x \in S$ can now be computed by a NC_2 circuit. To show this, we illustrate how to do so in nondeterministic log-space, since it is known that $NL \subseteq NC_2$:

> Starting from the initial id, until a final id is encountered, simulate M on x. When a beg-write-id α is encountered, nondeterministically guess a β of length $O(\log n)$. If the $\langle \alpha, \beta \rangle$th bit of U_K is 1, then let α be β and continue the simulation. If that bit is 0, then halt and reject.

All ids α and β are at most $O(\log n)$ in length, so this is an NL operation. Since $NL \subseteq NC_2$, this is also an NC_2 operation. To recap, the string U_K can be computed from x by a circuit of depth $O(\log^3 n)$ and polynomial size. From U_K and x, a circuit of depth $O(\log^2 n)$ can determine if $x \in S$. Therefore, $S \in NC_3^A$. ∎

The proof has actually shown something slightly stronger. If the sNSPACE machine being simulated has stack height bounded by $O(\log n)$ and questions of at

most polynomial length—thus using $O(\log^2 n)$ space—then the language it accepts is in NC_3^A. As mentioned earlier, if the *sNL* machine has constant stack height and asks polynomial length questions or has $O(\log n)$ stack height and asks constant length questions, then the language will be in NC_2^A. However, the circuit in general must be set up to accommodate many short questions on one input and few long questions on another.

4. Related Results

As further positive evidence of the applicability of this model, we see that Savitch's theorem relativizes.

THEOREM 4. *Let $f(n)$ be a space bound. Then for all oracles A,*

$$s\text{NSPACE}^A(f(n)) \subseteq s\text{SPACE}^A(f^2(n)).$$

Proof. For an arbitrary $L \in s\text{NSPACE}^A(f(n))$, let M be the nondeterministic Turing machine with an oracle stack operating under the RST restriction accepting L. And let x, $|x| = n$, be some input. Consider all ids of M on x for which the stack is empty. As in the unrelativized case, there are only $2^{O(f(n))}$ such ids. Suppose I_1 and I_2 are two such ids. If I_1 is a beg-write-id which deterministically leads to I_2, the next id where the stack is empty, we will view this as a *single* nondeterministic step. Note that it is only in such segments of the computation that the use of the oracle stack takes place.

So for two ids I_1 and I_2, I_1 can lead to I_2 in one nondeterministic step in two ways: either by a deterministic oracle query sequence as under the RST restriction or by I_1 causing a nondeterministic choice, one choice of which leads to I_2 in a single move (or, third, I_1 could equal I_2). In any case, the claim that I_1 leads to I_2 in one nondeterministic step could be verified in $f(n)$ space deterministically, possibly using the oracle stack.

To check if I_1 leads to I_2 in at most 2^i nondeterministic steps, we would test, for all empty stack ids I_{int}, whether I_1 leads to I_{int} and I_{int} leads to I_2, each within 2^{i-1} nondeterministic steps. This suggests the standard recursive algorithm [15; see also 8], the only modification being to the testing done at the base case ($i = 0$) where a slightly more complicated test may require some oracle calls, though still this can be done in $f(n)$ space. The depth of the recursion would be $O(f(n))$, and the number of bits needed to be saved at each level would be $f(n)$, the length of the description of an id. Hence, the deterministic space requirement would be $O(f^2(n))$. ∎

The oracle stack still does not allow one to easily separate *sL* from *sNL* relative to some oracle. In this sense, they behave much like relativized *L* and *NL*. This extends a result from [16].

THEOREM 5. *$L = NL$ if and only if, for all oracles A, $sL^A = sNL^A$.*

Proof. We can adapt a proof found in [12] to accomplish this. It suffices to prove that if $L = NL$, then for arbitrary A, $sNL^A \subseteq sL^A$. Noticing that any machine M accepting a set $S \in sNL^A$ has at most cn^k, for some c and k, ids on inputs x of length n, we define $\hat{M}$ as follows. $\hat{M}$ takes as input $x \# y_1 \# y_2 \# \cdots \# y_{cn^k}$, $|y_i| = O(\log n)$. $\hat{M}$ simulates M on input x, but when M, in id α, starts to write to the oracle stack, the computation resumes from id y_α. The set $\hat{S}$ accepted by $\hat{M}$ is in $NL = L$. So there is a deterministic Turing machine M_0 accepting this set. To accept the same set of strings as M, simulate M_0. When it needs a y_α, simulate the deterministic portion of M starting from id α. This will use $O(\log n)$ space on the oracle stack. When the stack is empty, pass the resultant id back to M_0. ∎

5. CONCLUSION

We have introduced a new measure of relativized space, that of the oracle stack. Using this measure, we saw that for all oracles A, both $NC_1^A \subseteq sL^A$ and $sNL^A \subseteq NC^A$. Previous space measures have failed to be faithful in this manner. The latter containment followed from the fact that $sNL^A \subseteq NC_3^A$, and we conjecture that $sNL^A \subseteq NC_2^A$. It was also seen that Savitch's theorem relativizes and that if one can construct an oracle A such that $sL^A \neq sNL^A$, then $L \neq NL$.

It seems a feature of this theory that as we desire more known results to relativize, we must continually modify the models. This is still an interesting thing to do, for each relativization will tell us something about the proofs involved in refining a particular containment. However, as the models get progressively more complicated, what we learn about the types of proofs is apt to become less general.

Note added in proof. Theorem 5 can easily be modified to show that

> $L = NL$ if and only if, for all oracles A and constructible functions $s(n) \geq \log n$, $s\text{SPACE}^A(s(n)) = s\text{NSPACE}^A(s(n))$.

To prove this, we adapt the proof given above. Assume that $L = NL$. After picking an $S \in s\text{NSPACE}^A(s(n))$ accepted by M, define $\hat{M}$ which takes input $x \# y_1 \# y_2 \# y_{2^{cs(n)}}$, $|y_i| = cs(n)$, c constant. $\hat{M}$ will simulate M on x, but when M in id α begins to write to the oracle stack, the simulation will resume from id y_α. Let $\hat{S}$ be the language accepted by $\hat{M}$. It is easy to see that $\hat{S}$ is in NL since $\log(n + cs(n)^{2^{cs(n)}}) \geq s(n)$, the space required by M. By assumption, then, $\hat{S}$ is in L and is accepted by some deterministic machine M_0. To see that $S \in s\text{SPACE}^A(s(n))$, we simulate M_0 on x, and when it needs to see part of y_α, simulate M on x from id α. This simulation will use space $O(\log(n + cs(n)\, 2^{cs(n)})) = O(s(n))$.

REFERENCES

1. D. ANGLUIN, "On Relativizing Auxiliary Pushdown Machines," *Math. Systems Theory* **13** (1980), 283–299.
2. T. BAKER, J. GILL, AND R. SOLOVAY, Relativizations of the $P = ?NP$ question, *SIAM J. Comput.* **4**, No. 4 (1975), 431–452.
3. T. BAKER AND A. SELMAN, A second step toward the polynomial hierarchy, *in* "Proceedings, 17th Found. of Comput. Sci., 1976, pp. 71–75.
4. A. BORODIN, On relating time and space to size and depth, *SIAM J. Comput.* **6**, No. 4 (1977), 733–744.
5. J. BUSS, Relativized alternation, *in* "Proceedings, Structure in Complexity Theory Conference," Lecture Notes in Computer Science Vol. 223, Springer-Verlag, New York/Berlin, 1986.
6. S. COOK, private communication.
7. S. COOK, A taxonomy of problems with fast parallel algorithms, *Inform. and Control* **64**, No. 1 (1985).
8. J. HOPCROFT AND J. ULLMAN, "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, MA, 1979.
9. R. LADNER AND N. LYNCH, Relativizations of questions about log-space reducibility, *Math. Systems Theory* **10** (1976), 19–32.
10. P. ORPONEN, General nonrelativizability results for parallel models of computation, *in* "Proceedings, Winter School on Theoretical Computer Science, 1984," pp. 194–205.
11. N. PIPPENGER, On simultaneous resource bounds (preliminary version), *in* "Proceedings, 20th Found. of Comput. Sci., 1979," 307–311.
12. C. RACKOFF AND J. SEIFERAS, Limitations on separating nondeterministic complexity classes, *SIAM J. Comput.* **10**, No. 4 (1981), 742–745.
13. W. RUZZO, On uniform circuit complexity, *J. Comput. System Sci.* **22**, No. 3 (1981), 365–383.
14. W. RUZZO, J. SIMON, AND M. TOMPA, Space-bounded hierarchies and probabilistic computations, *J. Computer System Sci.* **28**, No. 2 (1984), 216–230.
15. W. SAVITCH, Relationships between nondeterministic and deterministic tape complexities, *J. Computer System Sci.* **4**, No. 2 (1970), 177–192.
16. I. SIMON, "On some Subrecursive Reducibilities," Ph.D. dissertation, Stanford University, March 1977.
17. C. WILSON, Relativized circuit complexity, *J. Comput. System Sci.* **31**, No. 2 (1985), 169–181.
18. C. WILSON, Relativized NC, *Math. Systems Theory* **20** (1987), 13–29.
19. A. YAO, Separating the polynomial-time hierarchy by oracles, *in* "Proceedings, 26th Found. of Comput. Sci., 1985," pp. 1–10.